Name: _____ Andrew Id: _____

15-112 Fall 2024 Quiz 8

Up to 25 minutes. No calculators, no notes, no books, no computers. Show your work!
Do not use try/except or recursion on this quiz.

1. (4 points) **Code Tracing**: Indicate what the following program prints. Place your answer (and nothing else) in the box below the code.

```
def ct(d, lst):
    t = set()
    for e in lst:
        if e in d:
            t.add(e)
    a = d.keys() - t
    b = set(lst) - d.keys()
    print(f"t: {sorted(t)}")
    print(f"a: {sorted(a)}")
    print(f"b: {sorted(b)}")
    ret = []
    for e in b:
        for (k, v) in d.items():
            if e == v:
                ret.append(k)
    return sorted(ret)

a = {3: 2, 7: 8, 1: 15, 5: 2, 8: 6}
b = [1, 2, 3, 4, 5, 6]
print(ct(a, b))
```

2. (6 points) **Short Answer:** For each of the three functions shown below, write next to each line of the function either the Big-O runtime of the line or the number of times the line loops. Then write the total Big-O runtime of the function in terms of N in the box to the right of the code.

```
# lst1 & lst2 are lists of length N
def sa1(lst1, lst2):                                  # Big-O
    x = 0                                             # _____
    for i in range(len(lst1)):                        # _____
        if lst1[i] in lst2:                           # _____
            for j in range(len(lst2)-1, -1, -1):      # _____
                if lst1[i] == lst2[j]:                # _____
                    x += 1                            # _____
    return x                                          # _____
```

```
def sa2(lst): # lst is a list of length N             # Big-O
    if len(lst) == 0:                                 # _____
        return False                                  # _____
    if lst[0] != min(lst):                            # _____
        lst.sort()                                    # _____
    tmp = lst[::2]                                    # _____
    return max(lst) in tmp                            # _____
```

```
def sa3(s): # s is a string with N characters         # Big-O
    for letter in string.ascii_uppercase:             # _____
        if s[-1] == letter:                           # _____
            return ""                                 # _____
    i = len(s) - 1                                    # _____
    result = ""                                       # _____
    while i >= 0:                                     # _____
        result += s[int(i)]                           # _____
        i -= len(s) / 4                               # _____
    return result                                     # _____
```

3. (10 points) **Free Response**: Unique Sum Pairs

Write the function `findUniquePairs(L, target)` that takes a list of integers `L` and an integer `target` that returns a list of all the unique pairs of integers in `L` that sum up to `target`. Each pair should be represented as a tuple `(a, b)`. The order of the pairs in the returned list does not matter, nor does the order of the two numbers within each pair.

For example, consider the following testcase:

```
L = [1, 3, 2, 2, 4, 5, -1, 3, 1]
print(findUniquePairs(L, 1)) # This prints [(2, -1)]
print(findUniquePairs(L, 5)) # This prints [(3, 2), (1, 4)]
print(findUniquePairs(L, 4)) # This prints [(5, -1), (1, 3), (2, 2)]
```

The returned list should not contain duplicate pairs, even if the order is swapped. For example, the list with target sum of 4 could not include both `(1, 3)` and `(3, 1)`, as those are considered the same pair. It also does not contain any duplicates, so `(1, 3)` is only included once even though there are two different pairs of `(1, 3)` in the original list.

The straight-forward solution of checking each possible pair using nested loops is $O(N^2)$, but you'll need to do better than that. Your solution must be $O(N)$ where N is the length of `L`. (Solutions that are not $O(N)$ will receive significant deductions, but will still get some partial credit.)